

- Previously: best effort datagram - IP implementation
 - IP impl: (destination IP address, payload) -> Internet datagram {src IP address, dst address, TTL, payload, protocol}
 - Best-effort delivery
 - Delivered once
 - Never delivered
 - Delivered $n > 1$ times
 - Delivered with altered or truncated payload
 - Delivered to wrong destination
 - Delivered after another datagram that was sent later
 - User datagram protocol impl: (dst IP address, dst port #, payload)

(UDPSocket::sendto(...)) -> User datagram {contains src port, dst port, payload} (becomes payload of IP impl input)
 - Both provides best-effort delivery of datagrams ("unreliable")
 - The fundamental difference between UDP and IP is that UDP is port-to-port (between unprivileged user programs) and IP is host-to-host/computer-to-computer (entities have internet addresses).
 - UDP and IP are in the OS kernel, but whatever that is on the top of UDP can be in user-space. Unprivileged user programs would not affect each other if the OS assigns a different port address to each user programs.
 - Q: what if datagrams are altered during transmission?

A: UDP and IP have light checksum field for detecting alteration, and delivery to a wrong destination is something that could happen under "best-effort".
 - Q: how does Firefox know which dst port to send UDP to?

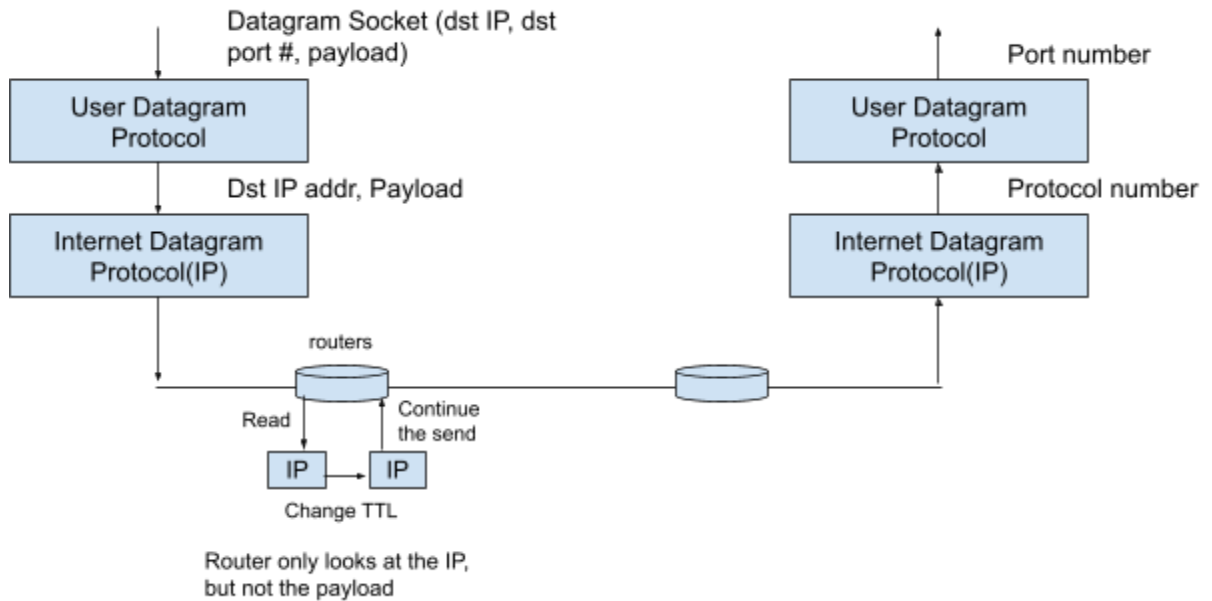
A: This is the same as how Firefox knows the dst IP address. (this question is deferred to later lecture)

(/etc/protocols: protocol index, /etc/services: port number)
 - Q: how does Firefox know its own port number?

A: The OS kernel assigns a src port to Firefox that won't conflict with any other user program.
 - Q: how does Firefox know the source address of incoming datagrams?

A: UDPSocket::recv: (src address &, payload &)
 - Q: Do I have to listen to the port as specified by /etc/services?

A: It is possible to run a service not on a well-assigned port number.
- The protocol stack is a sequence of modules where each module only communicates with the layer 1-level upper and lower from it through service abstraction.



-
- What can be built on top of UDP?
 - Unreliable text messaging
 - Voice over UDP (VoIP)
 - IP over UDP (VPN: Virtual Private Network)

Netflix -> UDP -> IP -> VPN Software -> UDP -> IP

Some machine in France: VPN Software <- UDP <- IP, and send the upper-layer IP datagram over the Internet.

When Netflix receives the datagram, it sees the src address as from France
- What if we want something more than “unreliable”? What if we want a “reliable retrievable short piece of data”?
 - Asker: request -> response can be built on top of UDP. Asker keeps sending until it gets an answer. Asker would eventually returns a response, and is “reliable”
 - `host cs144.keithw.org` is built as a user-program on top of UDP, and provides asker’s reliability (either returns or time-out)
 - For certain kind of requests, it is ok to make the request more than one time, but this is not the case for all requests.
 - What is the difference between reliable retrieval (“host cs144.keithw.org”) and reliable action (“Fire a torpedo”)? The kind of reliability that the service provides dictates how we design the service.